1 Wprowadzenie

Zanim przystąpimy do przedstawienia rozwiązania problemu poczyńmy kilka obserwacji. Po pierwsze, zastanówmy się kiedy linijka na strone w notatniku Jack'a nie może się przytrafić dla danej liczby pielgrzymów i danej kwoty w kieszeni Jack'a:

Stwierdzenie 1. *Linia **COLLECT** zawsze może się przydarzyć. Linia **PAY** jest niemożliwa tylko wtedy, gdy kwota do zapłacenia jest większa niż kwota w kieszeni Jacka. Linia **IN** lub **OUT** jest możliwa tylko gdy aktualny liczba euro jest podzielna przez liczbę osób w grupie przed tą linijką oraz (w wypadku linii **OUT**) jeśli liczba osób opuszczających grupę jest mniejsza niż liczba osób w grupie.*

Stwierdzenie 2. *Rozważmy pewien ciąg S linii w notatniku Jack'a. Załóżmy, że zanim pierwsza z tych linii została napisana było k pielgrzymów i m euro, a po tym, gdy wszystkie linie zostały napisane było k' pielgrzymów i m' euro. Wtedy, gdyby te same linie zostały napisane w sytuacji, gdy było k pielgrzymów i $m+kx$ euro na początku, to po napisaniu tych linii mielibyśmy k' pielgrzymów i $m' + k'x$ euro — zmiana liczby pieniędzy oczywiście nie zmienia liczby pielgrzymów, a zachowanie się kwot pieniędzy sprawdza się prostą indukcją.*

Możemy zatem zauważyć, że dodanie wielokrotności liczby pielgrzymów k w euro do kieszeni Jack'a nie psuje tego, czy k może być rozwiązaniem dla danej strony. Możemy więc założyć, że Jack ma dość pieniędzy by wykonać wszystkie opłaty od samego początku, a zatem linia **PAY** zawsze jest możliwa. Mamy też następujące spostrzeżenia:

Spostrzeżenie 3. *Wszystkie linie **COLLECT** na stronie można zaniedbać.*

Faktycznie, linie **COLLECT** jedynie dodają wielokrotność obecnej liczby pielgrzymów w euro do kieszeni Jack'a. Na mocy stwierdzenia 2 ta kwota (na osobę) będzie po prostu przenosiła się z linii do linii nie wpływając na to, czy linie **IN** oraz **OUT** są możliwe.

Spostrzeżenie 4. *Dowolny początkowy ciąg linii **PAY** można zaniedbać.*

Same w sobie początkowe linie **PAY** zawsze mogą wystąpić. Ich końcowym efektem jest odjęcie pewnej kwoty e z kieszeni Jack'a. Jeśli pozostała część strony jest możliwa z k pielgrzymami i m euro, to cała strona jest możliwa z k pielgrzymami i $m + e$ euro.

Spostrzeżenie 5. *Dowolny końcowy ciąg linii **PAY** można zaniedbać.*

Linie **PAY** zawsze same w sobie są możliwe, a stan, który po nich pozostanie jest nieistotny (bo nie ma po nich żadnych linii **IN** ani **OUT**).

Aby sprawdzić, czy dany ciąg linii może wystąpić będziemy używali następującego stwierdzenia:

Stwierdzenie 6. *Załóżmy, że część danych ma następującą postać:* ***IN/OUT*** a_1 ***PAY*** k_1 ***PAY*** k_2 $\ldots$ ***PAY*** k_n ***IN/OUT*** a_2 *oraz po tym jak* a_1 *pielgrzymów przyłączyło się (opuściło) do grupy, grupa liczyła dokładnie* x *osób. Z tego wynika, że* x *jest dzielnikiem* $k_1 + k_2 + \ldots + k_n$*. Ponadto dowolna wartość* x *o tej własności może być liczbą osób w grupie po wierszu* ***PAY*** k_n*.*

Istotnie, jeśli x nie byłby dzielnikiem $k_1 + k_2 + \ldots + k_n$ po tym jak a_2 osób przyłączyło się bądź opuściło grupę Jack musiałby natrafić na niecałkowitą liczbę euro, co jak wiemy nigdy nie nastąpiło.

Spostrzeżenie 7. *Jeli dany ciąg wpisów nie zawiera (po odrzuceniu* ***COLLECT*** *oraz początkowych i końcowych* ***PAY****) linii* ***PAY*** *(zawiera jedynie wpisy typu* ***IN*** *oraz* ***OUT****) oraz* $\Delta p_1, \Delta p_2, \ldots, \Delta p_n$ *jest ciągiem zmian liczby osób w grupie (*Δp_i *odpowiada* i*-temu wierszowi:* ***IN*** Δp_i *lub* ***OUT*** $-\Delta p_i$*), wtedy dowolna liczba osób nie mniejsza niż* $min_k - (\Delta p_1 + \Delta p_2 + \ldots + \Delta p_k)$ *może być początkową liczbą pielgrzymów.*

Mając na uwadze wszystkie powyższe spostrzeżenia oraz stwierdzenia możemy przedstawić następujący schemat rozwiązania:

1. Usuwamy wszystkie wpisy **COLLECT**.
2. Usuwamy wszystkie początkowe oraz kocowe wpisy typu **PAY** (jeśli takie występują).
3. Jeśli (po wykonaniu pierwszych 2 kroków) nie występują żadne wpisy typu **PAY** obliczamy minimalną liczbę pielgrzymów na podstawie obserwacji 7.
4. W przeciwnym wypadku mamy do czynienia z ciągiem składającym się z wpisów **IN/OUT** oraz **PAY**.
5. Weźmy pierwszą sekwencję informacji **PAY**, obliczmy ich sumę S.
6. Dla każdego dzielnika liczby S sprawdźmy, czy dzielnik ten może być liczbą osób w grupie w danym momencie, biorąc pod uwagę pozostałe wpisy.

Należy zaznaczyć, że powinniśmy znaleźć wszystkie dzielniki liczby S w czasie $O(\sqrt{S})$.

Ostatnią rzeczą jaką musimy umieć zrobić jest sprawdzenie, czy dana liczba m może stanowić liczność grupy biorąc pod uwagę pozostałe wpisy. Możemy tego dokonać wykonując prostą pętlę po wszystkich wpisach. Będziemy uaktualniać liczbę osób w grupie po każdym wpisie typu **IN/OUT** oraz dla każdego bloku informacji **PAY** będziemy obliczać ich sumę i sprawdzać czy jest ona podzielna przez aktualną liczbę osób w grupie (jeśli nie jest podzielna, to początkowa liczba osób w grupie była nieprawidłowa).

2 Rozwiązanie wzorcowe

Rozwiązanie wzorcowe (plik *pil.*{*cpp/java*}) jest implementacją powyższego algorytmu. Na początku usuwamy zbędne wpisy (opisane w punkcie 1) oraz 2)). Później wywołanie funkcji *no_pays()* obsługuję przypadek, gdy nie pozostał ani jeden wpis typu **PAY**. Następnie wywołana jest główna funkcja programu. Sumowane są tam wartości w pierwszym bloku operacji **PAY** (nazwijmy tą wartość *sum*), oraz dla każdej pary jej dzielników j oraz sum/j dodajemy je do wyniku, jeśli są to możliwe liczby osób w danym momencie w grupie. Odbywa się to za pomocą wywołania funkcji *possible()*, sprawdzając wcześniej, czy rozważana wartość jest nie mniejsza niż 1+ zmiana liczby osób w grupie spowodowana wpisem **IN/OUT**. Funkcja *possible()* implementuje schemat przedstawiony na kocu poprzedniej sekcji.

Niech N będzie rozmiarem danych. Złożoność obliczeniowa rozwiązania wzorcowego wynosi $O(N\sqrt{2000N})$, natomiast pamięciowa to $O(N)$.

3 Rozwiązanie wolne 1

Rozwiązanie wolne 1 (*pils1.*{*cpp/java*}) jest bardzo podobne do rozwiązania wzorcowego. Jedyną różnicą jest sposób znajdowania wszystkich dzielników liczby S poprzez pętlę po wszystkich liczbach naturalnych nie większych niż S, co zajmuje istotnie więcej czasu. Niech N będzie rozmiarem danych. Złożoność obliczeniowa wynosi $O(2000N^2)$, natomiast pamięciowa $O(N)$

4 Testy

Testy umieszczone są w następujących plikach:

- *pil0.in* — test przykładowy,

- *pil1.in* — prosty test poprawnościowy,
- *pil2.in* — bardziej złożony test poprawnościowy,
- *pil3.in* — test wydajnościowy składający się z krótkich (1-3 elementowych) początkowych i kocowych bloków **IN**/**OUT** oraz dużego (40-48 elementowych) bloku wpisów typu **PAY** ze stosunkowo dużymi kwotami (1850-2000). Zatem sumaryczna kwota jest wystarczająco duża aby odróżnić rozwiązanie wzorcowe od rozwiązania wolniejszego.
- *pil4.in* — losowy test wydajnościowy, składa się od z naprzemiennych bloków **IN**/**OUT** (najczęściej krótkich) oraz **PAY**. Pierwszy znaczący blok wpisów **PAY** jest stosunkowo długi, tak aby wszystkie dzielniki, które musimy znaleźć występowały w liczbie $50,000$-$70,000$.

Pierwsze dwa testy zostały stworzone ręcznie, pozostałe zostały wygenerowane w sposób automatyczny przez program *pilingen.cpp*, gdyż testy efektywnościowe powinny zawierać około $10,000$ przypadków testowych, co daje łączną liczbę $400,000$-$500,000$ wierszy. Ponieważ informacje **COLLECT** nie są istotne dla rozwiązania zadania, tylko kilka takich wpisów znajduje się w losowych miejscach ostatnich dwóch testów.